Raspberry IO Documentation

Release 0.1

Python Software Foundation

August 06, 2013

CONTENTS

Raspberry IO is an open source project. We welcome contributions. You can help by fixing bugs, planning new features, writing documentation, writing tests, or even managing the project. This section discusses some technical details of contributing to the Raspberry IO project.

CHAPTER

ONE

CONTRIBUTION DOCUMENTATION

1.1 Get the code

Here's how you can get the Raspberry IO site running on your computer.

1.1.1 Install Dependencies

Raspberry IO is a Django project built on top of the Mezzanine CMS, using PostgreSQL as our database. To get started, you will need the following programs installed. These should be installed using your operating system's standard package management system:

- Python >= 2.6 (2.7 recommended)
- pip >= 1.1
- virtualenv >= 1.7
- virtualenvwrapper >= 3.0
- PostgreSQL >= 8.4 (9.1 recommended)
- git >= 1.7

1.1.2 Clone the repository

Clone the Raspberry IO repository from Github:

```
$ git clone git@github.com:python/raspberryio.git
Cloning into 'raspberryio'...
remote: Counting objects: 3860, done.
remote: Compressing objects: 100% (1749/1749), done.
remote: Total 3860 (delta 2081), reused 3845 (delta 2069)
Receiving objects: 100% (3860/3860), 2.98 MiB | 861.00 KiB/s, done.
Resolving deltas: 100% (2081/2081), done.
```

You'll now have a new `raspberryio` subdirectory containing the code.

1.1.3 Set up your environment

Change into the `raspberryio` directory:

```
$ cd raspberryio
$
```

Create a virtual environment to work in and activate it:

```
$ mkvirtualenv --distribute raspberryio
...
$ pip install -r requirements/dev.txt
...
$
```

Create a local settings file and set your `DJANGO_SETTINGS_MODULE` to use it:

```
$ cp raspberryio/settings/local.example.py raspberryio/settings/local.py
$ echo "export DJANGO_SETTINGS_MODULE=raspberryio.settings.local" >> $VIRTUAL_ENV/bin/postactivate
$ echo "unset DJANGO_SETTINGS_MODULE" >> $VIRTUAL_ENV/bin/postdeactivate
$
```

Add the project directory to the virtualenv, deactivate and reactivate it to set up the environment variables above:

```
$ add2virtualenv .
$ deactivate
$ workon raspberryio
$
```

1.1.4 Set up the database

Create the Postgres database.

```
$ createdb -E UTF-8 raspberryio
```

Run the initial syncdb/migrate. When asked to create a superuser, answer `no`.

```
$ django-admin.py syncdb
$ django-admin.py migrate
```

> **Warning:** Creating a superuser in the syncdb step will trigger the error `django.db.utils.DatabaseError: relation "userprofile_profile" does not exist` because of a required one-to-one relation with a user profile model that doesn't exist in the database yet.

Now, create a superuser (This will also create the profile correctly):

```
$ django-admin.py createsuperuser
```

1.1.5 Master versus Develop branch

The `master` branch in the Raspberry IO repository represents the code that is running on the production site, raspberry.io. The staging server runs off the code on the `develop` branch. Most work should be done on feature branches off `develop` and then migrated to the `master` branch by the project maintainers, once it has been shown to be stable. We generally use the git flow model of development.

```
$ git checkout develop
Branch develop set up to track remote branch develop from origin.
Switched to a new branch 'develop'
$
```

1.1.6 Run the tests and server

Verify that everything is okay by running Raspberry IO's tests.

```
$ django-admin.py test
[lots of output omitted]
Ran 158 tests in 62.170s

OK
[lots of output omitted]
$
```

Then run the development server and play around!

```
$ django-admin.py runserver
```

1.2 Submit a pull request

To ask for a change you've made in your own Raspberry IO repository to be merged into the official repository, you make a pull request. If you're already familiar with Github and pull requests, just open a pull request against the `develop` branch of the Raspberry IO repository. (https://github.com/python/raspberryio/tree/develop) Be sure that your code passes all tests, conforms to our *code standards*, and has appropriate *documentation*.

Read the rest of this document for more details.

1.2.1 Branch

Work on a branch from the *develop* branch. Raspberry IO uses the Git Flow system to manage branches and releases. If you're using the gitflow tool, you can start your branch with:

```
$ git flow feature start NAME
Switched to a new branch 'feature/NAME'

Summary of actions:
- A new branch 'feature/NAME' was created, based on 'develop'
- You are now on branch 'feature/NAME'

Now, start committing on your feature. When done, use:

     git flow feature finish NAME

$
```

If you're not using the gitflow tool, create a new branch from develop and switch to it:

```
$ git branch feature/NAME develop
$ git checkout feature/NAME
Switched to branch 'feature/NAME'
$
```

1.2.2 Fork the repository

Login to your Github account, then go to the Raspberry IO repository on Github and create a fork by clicking the `Fork` button near the top right of the page.

1.2.3 Add a remote

We'll assume you already have a local clone of the repository that you've been working in, so all you need to do is add a new remote definition pointing to your new fork. You might choose to name your new remote definition using your Github username, to keep it distinguished from the remote pointing at the official repository.

```
$ git remote add username git@github.com:username/raspberryio.git
$ git remote -v
username    git@github.com:username/raspberryio.git (fetch)
username    git@github.com:username/raspberryio.git (push)
origin      git://github.com/raspberryio/raspberryio.git (fetch)
origin      git://github.com/raspberryio/raspberryio.git (push)
$
```

1.2.4 Push your change to your own Github repository

Your change needs to be on Github before you can open a pull request against the Raspberry IO code. Unless you have Raspberry IO commit privileges, you'll need to upload your change to your own fork of the repository.

Assuming your branch name is feature/NAME:

```
$ git push -u username feature/NAME
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:username/raspberryio.git
 * [new branch]      feature/NAME -> feature/NAME
Branch feature/NAME set up to track remote branch feature/NAME from username.
$
```

1.2.5 Open a pull request

Go to your fork of the Raspberryio repository on the Github web site (https://github.com/username/raspberryio/).

Click the `Pull request` button near the top center of the page.

Check the settings:

base repo python/raspberryio

base branch develop

head repo username/raspberryio (your repo)

head branch feature/NAME (your branch)

Enter an informative name and description for your pull request. By default, Github will try to pull these from your commit messages, but often you can improve them.

Your pull request description is your chance to convince people that your change is worthwhile and will benefit Raspberry IO. Here are some things to consider addressing:

- why is this change needed
- why will this be useful to general Raspberry IO users (not just you)
- what change was made
- why that change was made, as opposed to some other way of achieving the same ends
- what testing has been done
- etc.

Switch to the `Files Changed` tab and check that the changes shown are what you expect. If not, go back and check that you've committed the changes you intended on your branch, that you pushed it to your repo, and that the pull request settings are correct for your repo and branch.

When everything looks right, switch back to the `New Pull Request` tab and click the green `Send pull request` button in the lower right, below the description box.

1.3 Conform to best practices

We follow these practices when developing Raspberry IO code:

1. We generally follow the git flow model of development, which means most of your work will be on a feature branch off the `develop` branch.
2. Follow **PEP 8** style conventions. Use 4 spaces instead of tabs.
    - To learn more about writing 'pythonic' code, check out The Hitchhiker's Guide To Python's Code Style guide
    - Tip: Configure your development environment for python to make your life a bit easier.
3. Use CapitalizedCase for class names, underscored_words for method names.
4. Code using `os.path` must be Windows and 'NIX friendly. For example, don't use backslashes (`\`) as path separators.
5. Be sure every class and method has docstrings.
6. Use Python logging whenever an error or exception occurs. Optionally include debug-level logging.
7. Write a test which shows that the bug was fixed or that the feature works as expected.
8. Run the test suite to make sure nothing unexpected broke. We only accept pull requests with passing tests.
9. Write new or update existing *documentation* to describe the changes you made.
10. Add the change to the CHANGELOG for the **next** release.
11. Add your name to the AUTHORS file
12. Submit a pull request and get reviews before merging your changes, even if you have authority to merge the changes yourself.

1.4 Write documentation

We believe Raspberry IO needs to have documentation as good as our code. It's what you're reading now and is generally the first point of contact for new developers. We value great, well-written documentation and aim to improve it as often as possible. And we're always looking for help with documentation!

1.4.1 Getting the raw documentation

The official documentation is available on Read the Docs. This is the compiled HTML version. However, we edit it as a collection of text files (reST format) which live in the top-level `docs/` directory of a Raspberry IO release. Note that the official documentation is currently built from the `develop` branch.

1.4.2 Using Sphinx

Before building the documentation, you must have downloaded the Raspberry IO source code. See the *Get the code* section for instructions on installing Raspberry IO locally.

We use the Sphinx documentation system (based on docutils). To build the documentation locally, you'll need to install Sphinx.

```
$ pip install Sphinx
```

Then, building the HTML is easy. Just run make from the `docs` directory.

```
$ cd raspberryio/docs
$ make html
```

(or `make.bat html` on Windows)

The compiled HTML code will be in `raspberryio/docs/_build/`.

To get started contributing, you'll want to read the reStructuredText Primer. After that, you'll want to read about the Sphinx-specific markup that's used to manage metadata, indexing, and cross-references.

1.4.3 Documentation starting points

Typically, documentation changes come in two forms:

- **General improvements:** typo corrections, error fixes and better explanations through clearer writing and more examples.
- **New features:** documentation of features that have been added to the framework since the last release.

If you're interested in helping out, a good starting point is with the documentation label on the GitHub issue tracker.

1.5 Deployment

This section documents the process by which we deploy Raspberry IO. This is done on a staging site first, and then to a production site once the code is proven to be stable and working on staging.

Note: The deployment uses SSH with agent forwarding so you'll need to enable agent forwarding if it is not already by adding `ForwardAgent yes` to your SSH config.

1.5.1 Server Provisioning

The first step in creating a new server is to create users on the remote server. You will need root user access with passwordless sudo. How you specify this user will vary based on the hosting provider.

1. For each developer, put a file in the `conf/users` directory containing their public ssh key, and named exactly the same as the user to create on the server, which should be the same as the userid on the local development system. (E.g. for user `dickens`, the filename must be `dickens`, not `dickens.pub` or `user_dickens`.)
2. Run this command to create users on the server:

   ```
   fab -H <fresh-server-ip> -u <root-user> create_users
   ```

   This will create a project user and users for all the developers.

3. Lock down SSH connections: disable password login and move the default port from 22 to `env.ssh_port`:

   ```
   fab -H <fresh-server-ip> configure_ssh
   ```

4. Add the IP to the appropriate environment function and provision it for its role. You can provision a new server with the `setup_server` fab command. It takes a list of roles for this server ('app', 'db', 'lb') or you can say 'all'. The name of the environment can now be used in fab commands (such as production, staging, and so on.) To setup a server with all roles use:

   ```
   fab staging setup_server:all
   ```

5. Deploy the latest code to the newly setup server:

   ```
   fab staging deploy
   ```

6. If a new database is desired for this environment, use `syncdb`:

   ```
   fab staging syncdb
   ```

7. Otherwise, a database can be moved to the new environment using `get_db_dump` and `load_db_dump` as in the following example:

   ```
   fab production get_db_dump
   fab staging load_db_dump:production.sql
   ```

1.5.2 Vagrant Testing

You can test the provisioning/deployment using Vagrant. Using the Vagrantfile you can start up the VM. This uses the `precise64` box:

```
vagrant up
```

With the VM up and running, you can create the necessary users. Put the developers' keys in `conf/users` as before, then use these commands to create the users. The location of the vagrant key file might be:

```
if gem installed: /usr/lib/ruby/gems/1.8/gems/vagrant-1.0.2/keys/vagrant
if apt-get installed: /usr/share/vagrant/keys/vagrant
```

This may vary on your system. Running `locate keys/vagrant` might help find it. Use the full path to the keys/vagrant file as the value in the `-i` option:

```
fab -H 33.33.33.10 -u vagrant -i /usr/share/vagrant/keys/vagrant create_users
fab vagrant setup_server:all
fab vagrant deploy
fab vagrant syncdb
```

When prompted, do not make a superuser during the syncdb, but do make a site. To make a superuser, you'll need to run:

```
fab vagrant manage_run:createsuperuser
```

It is not necessary to reconfigure the SSH settings on the vagrant box.

The vagrant box forwards port 80 in the VM to port 8080 on the host box. You can view the site by visiting localhost:8080 in your browser.

You may also want to add:

```
33.33.33.10 vagrant.raspberry.io
```

to your hosts (`/etc/hosts`) file.

You can stop the VM with `vagrant halt` and destroy the box completely to retest the provisioning with `vagrant destroy`.

For more information please review the Vagrant documentation.

1.5.3 Deployment

For future deployments, you can deploy changes to a particular environment with the `deploy` command. This takes an optional branch name to deploy. If the branch is not given, it will use the default branch defined for this environment in `env.branch`:

```
fab staging deploy
fab staging deploy:new-feature
```

New requirements or South migrations are detected by parsing the VCS changes and will be installed/run automatically.

1.5.4 Releases

In general, every deployment to master should be a new released version of Raspberry IO. Currently hotfixes are an exception to this rule. Here's the steps involved in creating a new release. Let's assume that master is running version 0.1 and we have made a bunch of changes on the `develop` branch that we want to release as version 0.2:

1. Update `docs/CHANGELOG.rst` on `develop` branch and replace "(unreleased)" with "(today's date)"
2. Update `setup.py` on `develop` branch and change version to the new number (e.g. 0.2)
3. Deploy `develop` to staging, ensuring that everything works
4. Merge `develop` into `master`
5. Tag `master` on github as v0.2
6. Deploy `master` to production
7. Update `docs/CHANGELOG.rst` on `develop` branch, adding a new heading at the top: "Version 0.3 - (Unreleased)"
8. All new changes on `develop` should now be documented under the Version 0.3 heading in the CHANGELOG

1.5.5 SSL

Raspberry IO sits behind the PSF Load Balancer. Requests come in to the load balancer via either HTTP or HTTPS. The load balancer terminates the SSL connection and then forwards the request to our Django server with the `X-Forwarded-Proto` header set to either `http` or `https`. Django checks that header and sets `request.is_secure()` appropriately. Mezzanine routes any URLs beginning with `/account` or `/admin` to HTTPS. This can be configured within Mezzanine if other URL patterns need to be secure.

The Chef recipes which control the load balancer are located at https://github.com/python/psf-chef/tree/master/cookbooks/psf-loadbalancer

1.6 Raspberry IO Changelog

1.6.1 Version 0.2 - Open Source Release (2013-08-06)

- Set up SSL
- Transfer repo to PSF
- Set up Travis CI
- Set up Spinx docs, Read The Docs integration
- Initial version of 'How to Contribute' Documentation
- Revamped README
- Licensed under Apache 2.0

1.6.2 Version 0.1 (2013-07-15)

- #152: Fix pagination on project list
- #145: Fix RSS link on '/community' page
- #142: Fix 'most active users' calculation
- #147: Removed sitemap app
- #148: Added DEFAULT_FROM_EMAIL for password reset emails
- Initial release for Pycon 2013

CHAPTER

TWO

DEVELOPER DOCUMENTATION

2.1 Developer Documentation

Raspberry IO is a site to help users share knowledge about using the Python language on the Raspberry Pi platform. If you've gotten to this page, then you are interested in helping with the development of the site. Thanks! First, please check out the *contribution docs*, which explains how to get the code running and how to make sure your contributions meet our standards.

This document will give a high level overview of the site and its components. There are 4 main areas to the site, which are linked in the footer of the site:

Create This is where you share information about Raspberry Pi projects you create.

Explore This is where you can browse projects that community members have created.

Learn This is a wiki where you can learn about the Raspberry Pi and contribute your own knowledge to the community.

Community This includes a RSS aggregator which collects interesting Raspberry Pi posts and a Q&A forum where you can ask questions and help other users.

In addition, the site features social features where you can see information about other users, follow them, and view how they are interacting with the site.

2.1.1 Detailed info

The 'Create' and 'Explore' sections are built with the `raspberryio.project` app. It allows users to create projects with arbitrary numbers of steps (think steps in a tutorial), images, video links, etc. Projects can be labeled as 'Featured' by site admins. Projects can be in draft or published state.

The 'Learn' section is a wiki. We use a fork of django-wiki. The main reason for the fork is to use the Pillow imaging library rather than PIL, as Pillow installation has been simpler in our experience. We also heavily modified the templates.

As mentioned above, the 'Community' section includes two pieces of functionality. The Feed aggregator is in `raspberryio.aggregator`. This app allows users to submit RSS feeds and admins to approve or deny them. Feeds are then displayed to the community. The django-push app is used to manage the PubSub process.

The second portion is a Q&A forum which is included in the app `raspberryio.qanda`. This datamodel includes questions and answers, and allows users to upvote good answers. This is again built on top of Mezzanine models.

The social aspect is built by various packages. The `raspberryio.userprofile` app collects information about users. The django-activity-stream app is used to allow people to follow the activity of other users.

2.1.2 Search:

Search is implemented through the Mezzanine search facilities. We specify which models and fields we are interested in indexing in the `SEARCH_MODEL_INDEXES` dictionary in the base.py settings file. Proxy models which subclass Searchable are dynamically created in the `raspberryio.search` app, containing only those fields with the desired weights. (Fields with higher weights will be shown to the user before those with lower weights).

Searching the wiki is a special case. Since each revision of the wiki is saved, we could get multiple revisions of the same article show up in search results. So, instead of feeding the wiki.ArticleRevision model to Mezzanine, we have created an app `raspberryio.search_models` which extracts only the latest revision of each article and indexes that. The `search_models` app maintains this flattened representation of the latest revision in its own table, and should be used in the future for any other models with a similar structure. Using something in MPTT or similar might be better than the current "flatten and copy" behavior, but this is what we have for now.